NIT-249
NT0233US

Title of the Invention

METHOD AND SYSTEM FOR MANAGING DOCUMENTS

Inventors

Yukie KANIE
Toru TAKAHASHI
Yuki AOYAMA
Yasuki ITOU

TITLE OF THE INVENTION

Method and System for Managing Documents

BACKGROUND OF THE INVENTION

The present invention relates to a method and a system for managing documents so as to indicate changes among a plurality of versions of a document by specifying and showing each difference between the versions.

Recently, documents to be widely distributed and reused, such as product manuals, have been written and stored in structured document formats, such as SGML (Standard Generalized Markup Language) in order to facilitate their distribution and reuse. As for frequently revised documents, a new version document (hereinafter simply referred to as a version) is generated each time a document is revised, and each new version is generally managed as a separate file. To intuitively grasp changes between versions, it is effective to employ a "difference indication" method in which matching relationships between corresponding character strings in two files of different versions are extracted, and then portions (strings) which do not have corresponding matched portions (strings) are indicated as differences.

Japanese Laid-Open Patent Publication No. 9-319632 (1997) discloses a method for managing versions of structured documents, describing a technique of extracting differences between structured documents and indicating the extracted differences. Specifically, this version management method sets one of two versions written in a structured document format as a reference version, extracts difference information between the two versions (that is, extracts each change, and the portion to which the change has been made), and outputs an SGML document (hereinafter referred to as a "difference-embedded document"), which is structurally described and embedded in the reference version. This difference-embedded document can be displayed by use of SGML document editing software available, etc. to highlight the changes from the reference version. Furthermore, each version can be restored by interpreting the structural description of the changes (difference information) embedded in the difference-embedded document, and converting the structure of the difference-embedded document based on the structural description. That is to say, a difference-embedded document is a document in which contents of two versions are efficiently described so that the contents of two versions are separable.

Further, in order to efficiently manage versions of a frequently revised documents, the method for managing versions according to the above patent publication sets a certain version as a reference version. And each time a new version is created, the method extracts difference information between the new version and the reference version. After that, the method outputs a "difference document" in which only the obtained difference information is structurally described, and stores the output difference document as version management data in order to reduce the amount of data required for version management. In this case, it is possible to restore the two versions by interpreting the structural description of the changes (difference information) in each difference document, and converting the structure of the reference version based on the structural description.

Like HTML (Hyper Text Markup Language), XML (eXtensible Markup Language) is a structured document description language intended to be used on the Internet. XML is structurally a subset of SGML and can define a document structure freely as is the case with SGML. Not only can XML document data be displayed and printed out by use of an XML-aware Web browser, but also it can express various data based on a document structure defined for a specific application, which makes XML useful as a data exchange format on the Internet. Recently, various industries have been employing their industry-standard data exchange formats defined by use of XML.

A document body having a logical structure can be described in XML without using a prepared DTD (document type definition), which is not possible with SGML. However, it is necessary to mark up each element constituting the logical structure, instead, by sandwiching the element between a start tag and an end tag. To express an element "participant name" which is composed of elements "surname" and "first name", for example, it is necessary to write a line such as: "<participant name><surname>Hitachi</surname><first name>Taro</first name></participant name>". In SGML, on the other hand, if a DTD clarifies that the element "participant name" is composed of the elements "surname" and "first name", the end tags "</surname>" and "</first name>" can be omitted by writing a line such as: "<participant name><surname>Hitachi<first name>Taro</participant name>".

In XML, it is possible to write a document having a logical structure without necessarily preparing a DTD by sandwiching each element in the document between a start tag and an end tag, as described above. Capitalizing on this advantage, it is possible to freely combine various tag sets used for writing industry-standard data to write a document. The XML namespace is used to avoid "collision" between element names (duplication of an element name), which may occur when a plurality of tag sets are used in a document. Consider a case in which a document including formulas and tables is written using three types of tag sets for writing the entire document body, the formulas, and the tables, respectively. Furthermore, suppose that the document body tag set includes a tag "<title>" for indicating a document title, and the table tag set also includes a tag "<title>" for indicating a table title. In such a case, the expression "<title>XXX survey results</title>" appearing in the document, for example, is vague as to which tag set the expression belongs to. To clarify which tag set each tag belongs to, a tag indicating a table title, for example, is expressed as "<table: title>". In this case, the word "table" indicates a namespace specifying a table tag set. Use of namespaces enables an application to discriminate each tag set even in an XML document including a plurality of tag sets. Furthermore, it is possible to regard only a tag set belonging to a specific namespace as a target for processing.

SUMMARY OF THE INVENTION

The following description exemplifies a case in which an application program is maintained up to date by referring to descriptions of APIs (Application Programming Interfaces) in versions of a programmer's guide written in a structured document format. It is assumed that a person in charge of the maintenance knows beforehand the version number of the programmer's guide corresponding to the APIs currently used in the program to be maintained (this version number is hereinafter referred to as "the earlier version number", or "the version number of the earlier version"), and obtains changes made to the APIs by later versions to modify the program based on the changes.

To maintain an application program as described above, all APIs changed after the earlier version are extracted, and in which version (that is, when) each API change was made is specified. After that, it is necessary to determine the history of the changes indicating how changes were made on each version and obtain the contents of the latest version.

However, the conventional method for indicating difference information displays difference information obtained as a result of comparing only two versions: a reference version and each target version. Accordingly, when an earlier version is regarded as a reference version, and compared with the latest version to indicate their difference, the comparison results obtained indicate only the accumulated total difference (changes) between the reference version and the latest version. That is, it is not possible to obtain the information of how changes were made on each version between the reference and the latest versions. It may be possible to compare each newly revised version with its immediately previous version and extract the difference between them with regarding the immediately previous one as a reference version. However, when there are one or more versions generated between the earlier and the latest versions, it is necessary to trace the API change history from the earlier version to the latest version by checking changes between each version and its immediately previous version in between, and the portions to which the changes were made, resulting in time-consuming work.

A conventional difference-embedded document efficiently expresses the contents of two versions in a single document by using difference information derived from comparing the reference version and the target version. However, it is not possible to express the contents of more than two versions in one conventional difference-embedded document.

In order to solve the above problems, it is an object of the present invention to provide a method capable of expressing changes made to each version of a multi-version document.

To achieve the above object, a method of the present invention for generating a "multi-version" document using a revised version and its immediately previous version of a structured document extracts: common portions each composed of a common structure and text character strings included in both the revised version and its immediately previous version of the structured document; difference portions each specific to either the revised version or its immediately previous version; and correlation between structures included in the revised version and structures included in its immediately previous version, and correlation between text character strings included in the revised version and those in its immediately previous version.

This method then creates the following multi-version document. In this multi-version document, each extracted common portion has a pair of version description tags attached thereto one of which includes the version identifiers of the revised version and its immediately previous version, and the node identifiers of this extracted common portion in the revised version and its immediately previous version. This multi-version document further includes each extracted difference portion having a pair of version description tags attached thereto one of which includes the version identifier of the version including this extracted difference portion, the node identifier of the extracted difference portion in the version including the extracted difference portion, and information indicating whether the extracted difference portion has been produced as a result of insertion or modification. This multi-version document further includes the correlation information described above. Thus, the above method of the present invention creates a multi-version document in which contents of two versions are efficiently expressed.

When a newly revised version has been created, the last version included in the multi-version document and the newly revised version are subjected to the above processing to produce a new multi-version document.

Based on this multi-version document, it is possible to display any arbitrary version. Furthermore, difference between a specific version and another version can be displayed also based on this multi-version document.

As described above, the present invention is capable of expressing changes made to a specific version selected from among a plurality of versions, using a single multi-version document.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram showing the configuration of a system according to the present invention;

Fig. 2 is a schematic diagram showing processes employed by the present invention;

Fig. 3 is a diagram showing an example of structured document data (a programmer's guide V1) to be processed according to the present invention;

Fig. 4 is a diagram showing another example of structured document data (a programmer's guide V2) to be processed according to the present invention;

Fig. 5 is a diagram showing still another example of structured document data (a programmer's guide V3) to be processed according to the present invention;

Fig. 6 is a diagram showing a document tree of the programmer's guide V1;

Fig. 7 is a diagram showing an example of data used for difference extraction;

Fig. 8 is a diagram showing difference extraction results between the programmer's guides V1 and V2;

Fig. 9 is a diagram showing a correlation table for the programmer's guide V1;

Fig. 10 is a diagram showing a correlation table for the programmer's guide V2;

Fig. 11 is a problem analysis diagram (PAD diagram) of a program for generating multi-version documents;

Fig. 12 is a PAD diagram of a data normalization program;

Fig. 13 is a diagram showing normalized data of the programmer’s guide V1;

Fig. 14 is a diagram showing a multi-version document expressing the contents of the programmer’s guides V1 and V2;

Fig. 15 is a diagram showing a tag-attached version V2a;

Fig. 16 is a diagram showing difference extraction results between the tag-attached version V2a and the programmer’s guide V3;

Fig. 17 is a PAD diagram of a program for updating multi-version documents;

Fig. 18 is a diagram showing multi-version document data expressing the contents of the programmer’s guides V1, V2, and V3;

Fig. 19 is a PAD diagram of a program for outputting a specific version;

Fig. 20 is a PAD diagram of a program for displaying multi-version documents;

Fig. 21 is a diagram showing a difference/modification-history display example of a multi-version document;

Fig. 22 is a diagram showing a text cross-reference display example of a multi-version document;

Fig. 23 is a diagram showing a tree cross-reference display example of a multi-version document;

Fig. 24 is a schematic diagram showing processes employed by a second embodiment of the present invention;

Fig. 25 is a diagram showing an example of structured document data (English version) to be processed according to the second embodiment of the present invention;

Fig. 26 is a diagram showing another example of structured document data (Japanese version) to be processed according to the second embodiment of the present invention;

Fig. 27 is a diagram showing a correlation table for the English version;

Fig. 28 is a diagram showing a correlation table for the Japanese version; and

Fig. 29 is a diagram showing a multilingual document expressing the contents of the English version and the Japanese version.

DETAILED DESCRIPTION OF THE INVENTION

A first embodiment of the present invention will be described below with reference to the accompanying drawings.

Fig. 1 shows the system configuration of the first embodiment. In the figure, a computer 1 connected to a network 8 is a document management server having a difference extraction function for structured documents while a computer 2 is a client which obtains document data stored in the document management server through the network. Even though the present invention does not specify a communication means between the server and the client, the following description of the first embodiment assumes that the HTTP protocol on the Internet is used as the communication means between them. More specifically, it is assumed that the document management server 1 has an HTTP server function, and the client 2 communicates with the document management server 1 using a Web browser. Accordingly, the document management server 1 comprises a display 3, a data input device 4 such as a keyboard, a CPU 5, a memory 6, and a database 7 storing structured documents written in SGML, HTML, XML, etc. The memory 6 holds a difference extraction program 61 for structured documents, a program 62 for generating multi-version documents, a program 63 for updating multi-version documents, a program 65 for outputting a specific version, an HTTP server program 66, and a correlation table 67 generated by the difference extraction program 61 for structured documents. On the other hand, the client 2 comprises a display 9, an input device 10, a CPU 11, and a memory 12. The memory 12 holds a Web browser program 121, a program 64 for displaying multi-version documents, and the program 65 for outputting a specific version.

Fig. 2 schematically shows processes employed by this embodiment.

The process of creating a new multi-version document proceeds as follows. First, the difference extraction program 61 for structured documents is executed on two versions (for example, an original version 20 and a revised version 22) written in a structured document format. The difference extraction results are stored in the correlation table 67. Then, the program 62 for generating multi-version documents is executed to generate a multi-version document 24 of the XML format which expresses the contents of the two versions in a file, based on the difference extraction results stored in the correlation table 67.

The process of updating a multi-version document proceeds as follows. First, the program 65 for outputting a specific version is executed to output a latest version 262 as a specified version included in the multi-version document 24. Next, the difference extraction program 61 for structured documents is executed to compare the version 262 with a version to be added (for example, a further-revised version 30). The difference extraction results are stored in a correlation table 672. Then, the program 63 for updating multi-version documents is executed to generate a multi-version document 242 to which the contents of the additional version has been added, based on the difference extraction results stored in the correlation table 672.

In the client 2, on the other hand, the program 64 for displaying multi-version documents performs multi-version document display controls such as switching of versions to be displayed, and cross-reference display, while the program 65 for outputting a specific version performs output control of multi-version documents.

Description will be made of structured document examples used for newly creating or updating a multi-version document. Figs. 3, 4, and 5 shows programmer's guides V1, V2, and V3 written in a structured document format as document examples which correspond to the original version 20, the revised version 22, and the further-revised version 30 in Fig. 2, respectively. The underlined portions in Figs. 3 and 4 indicate portions common to both the version V1 and the version V2, while the italicized portions in Figs. 4 and 5 indicate portions common to both the version V2 and the version V3. A tree structure (referred to as a "document tree") as shown in Fig. 6 is generated by interpreting the structure of the structured document data shown in Fig. 3. A node ID is assigned to each node to uniquely identify every node included in this document tree. In this embodiment, the node ID of the route node is set to "1", and a node ID, such as "1_1" or "1_2", obtained as a result of adding an additional number to the node ID of a parent node is assigned to its child node or a node lower than that. In Fig. 6, the node ID of each node of the document tree is indicated to the lower left of the node.

Description will be made of the process performed on the versions V1 and V2 by the difference extraction program 61 for structured documents.

First, data of the versions V1 and V2 is converted to data suitable for difference extraction processing. To make it easy to compare character strings in the difference extraction process, this embodiment replaces each tag included in a document by a symbol as shown in Fig. 7.

Next, a difference extraction process is performed on the above converted data for difference extraction processing by using a method of extracting difference between structured documents, disclosed in Japanese Laid-Open Patent Publication No. 8-329079 (1996). Difference extraction results as shown in Fig. 8 are obtained. In the figure, for example, the character string "int exec_diff(char* fname1, char* fname2)" of the node 1_3 of the version V1 is divided into five portions such as a common portion "int exec_diff (char * ", a difference portion "fname1", another common portion ", char * ", another difference portion "fname2", and, finally, still another common portion “)”, for classification. These divided portions are hereinafter referred to as divided character strings. Furthermore, each divided character string is given a divided node ID obtained by adding an additional number to the node ID of the node including the divided character string. For example, the five divided character strings of the above node 1_3 are given divided node IDs “1_3_1”, “1_3_2”, “1_3_3”, “1_3_4”, and “1_3_5”, respectively in the order of their occurrence.

Then, based on coincidence of two divided character strings extracted as common portions, the matching relationship between a pair of nodes each including the respective one of the two divided character strings is extracted. For example, a matching relationship between the node 1_3 of the version V1 and the node 1_3 of the version V2, each node including the respective 1_3_1 node is derived from the coincidence between the divided character string “int exec_diff (char * ” of the divided node 1_3_1 of the version V1 and the divided character string “int exec_diff (char*” of the divided node 1_3_1 of the version V2.

After that, relationships between difference portions included in two nodes which have a matching relationship are extracted. For example, the above node 1_3 of the version V1 includes the divided nodes 1_3_1 and

1_3_3, which are portions common to the version V2, and the divided node 1_3_2 ("fname1"), which is a different portion positioned between the two common portions (the divided nodes 1_3_1 and 1_3_3), whereas the node 1_3 of the version V2 includes the divided nodes 1_3_1 and 1_3_3, which are portions common to the version V1, and the divided node 1_3_2 ("str1"), which is a different portion positioned between the common portions (the divided nodes 1_3_1 and 1_3_3). Therefore, the divided nodes 1_3_2 in the versions V1 and V2 each occupy the same position in the respective version. Since these two difference portions both include the character "1", it is determined that the two difference portions correspond to each other, and therefore one difference portion ("fname1" of the version V1) has been changed to the other ("str1" of the version V2) between the two versions V1 and V2. On the other hand, a divided node having no matching relationship between versions, such as the divided node 1_4_1_2 ", then" of the version V1 or the divided node 1_4_1_2 "." of the version V2 is regarded as a deletion or an insertion, respectively.

Fig. 9 shows a correlation table 67-1 storing difference extraction results obtained from the version V1 as a result of the above processing, while Fig. 10 shows a correlation table 67-2 storing difference extraction results obtained from the version V2.

Fig. 11 shows a flowchart of the process performed by the program for generating multi-version documents.

The following description assumes that a multi-version document is to be generated using the above versions V1 and V2.

First, a data normalization process is performed to normalize the contents of the version V1 based on the difference extraction results stored in the correlation table 67-1 at step 6202. Fig. 13 shows the normalized contents of the version V1. In the figure, all tags and text character strings included in the structured document are sandwiched between each pair of version description tags in units of common portions or difference portions. A pair of version description tags are used to discriminate each piece of data gathered and grouped for each version in a multi-version document. The pair of version description tags have attributes which hold version information and other information on data sandwiched by the pair of version description tags. This embodiment uses a type of version description tag to which the XML namespace is applied; specifically the tags starting with "<diff:ver" in Fig. 13. For example, the text character string of the divided node 1_4_1_1 of the version V1 is expressed as "<diff:ver vnum='1' node ID='1_4_1_1'>It executes the difference comparison processing</diff:ver>". In the version description tag, "vnum" indicates the version attribute while "nodeID" indicates the node attribute. The entry line "<function-list xmlns:diff='http://www.xxx.yyy/diff/'>" at the top of Fig. 13 is a namespace declaration which declares that the namespace "diff" is applied to all structures following the phrase "function-list". With this namespace declaration, it is possible to embed version description tags in a structured document having any structure without worrying about duplication of tag names. It should be noted that only processing systems which know the namespace "diff" can process the above version description tags.

Next, based on information on the common portions included in the correlation table 67-1, a version number of 2 is added to the version attribute of each pair of version description tags parenthesizing a common portion in the normalized data, and the value of the "corresponding divided node ID" field 6712 is added to the node attribute at step 6206. For example, since the divided node 1_4_1_1 of the version V1 is a common portion, the entry line of the divided node 1_4_1_1 of the version V1 is rewritten by an entry line such as: "<diff:ver vnum='1 2' nodeID='1_4_1_1 1_4_1_1'> It executes the difference comparison processing</diff:ver>".

Then, based on information on the difference portions included in the correlation table 67-2, new difference portion structures are generated at step 6210 by performing steps of: parenthesizing each difference portion with a pair of version description tags; setting its version attribute to the version number "2"; setting its node attribute to the value of the divided node ID field 6722; and setting its difference class attribute to the value of the difference type field 1209. For example, with respect to the difference portion "to the user-specified file" (the divided node 1_4_2_2) belonging to the node 1_4_2 of the version V2, a new difference portion structure (line) "<diff:ver vnum='2' nodeID='1_4_2_2' desc='insertion'>to the user-specified file</diff:ver>" is generated.

After that, the above generated new difference portion structures are inserted at proper positions determined based on relationships between the nodes included in the normalized data at step 6212. For example, it can be seen from the values of the "corresponding divided node ID" fields **1212** in the correlation table 67-2 that the above difference portion "to the user-specified file" belonging to the node 1_4_2 of the version V2 is inserted between the divided nodes 1_4_1_3 and 1_4_1_4 in the version V1. Therefore, its new difference portion structure is inserted at a position immediately before the version description tag whose version attribute and node attribute are "1" and "1_4_1_4" respectively in the normalized data.

A multi-version document as shown in Fig. 14 is generated by the above process. It may not be necessary to strictly manage node relationships between different versions depending on the type of a multi-version document to be created. If it is not necessary to strictly manage the node relationships, the node attribute and the difference class attribute may be omitted. For further simplification, version description tags common to both versions V1 and V2 may be omitted.

Fig. 12 shows a flowchart of the process performed by the data normalization program.

The following description assumes that the contents of the version V1 is to be normalized based on difference extraction results between the versions V1 and V2.

If the process of newly creating a multi-version document is to be performed (step 62022), all tags and text character strings included in the document are parenthesized with each pair of version description tags, and its version attribute is set to "1" at step 62024. Furthermore, a namespace declaration concerning version description tags is inserted with respect to the tag corresponding to the route node of the document tree at step 62025. With this, the above entry line "<para>It executes the difference comparison processing, then outputs the result.</para>" of the node 1_4_1 of the version V1 is rewritten by the entry line "<diff:ver vnum='1'><para><diff:ver vnum='1'> It executes the difference comparison processing, then outputs the result.</diff:ver></para></diff:ver>", and the start tag "<function_list>" of the route node is also rewritten by the entry line "<function_list xmlns:diff='http://www.xxx.yyy/diff/'>".

Next, referring to relationships between nodes included in the correlation table 67-1 (indicated by the node ID field 6711 and the "corresponding node ID" field 6718), it is determined whether, on each node belonging to the normalization-target version (in this case, the version V1) (62026), it matches two or more nodes of the other version (in this case, the version V2) at step 62028. If there is a node of the version V1 which is associated with two or more nodes of the version V2, the normalized entry line of the node is divided into the corresponding node Ys with the attribute of structure XX including the parenthesizing version description tags unchanged at step 62030. For example, the node 1_4_1 of the version V1 is associated with the two nodes 1_4_1 and 1_4_2 of the version V2 as shown in Fig. 9.

Therefore, the above normalized entry line (the node 1_4_1 of the version V1) is replaced by the two normalized entry lines "<diff:ver vnum='1'><para><diff:ver vnum='1'>It executes the difference comparison processing</diff:ver></para></diff:ver>" and "<diff:ver vnum='1'><para><diff:ver vnum='1'>, then outputs the result.</diff:ver></para></diff:ver>", which are obtained as a result of dividing the above original normalized entry line at a position between the nodes 1_4_1 and 1_4_2.

Next, on the common portion and the difference portion relating to the version to be normalized in the correlation table 67-1 (62032), if the common portion and the difference portion coincide partially with the structure A parenthesized by version description tags wherein the structure A includes the version to be normalized as a version attribute (62036), they are divided into a coincided portion and the other, with the attribute of the structure AA unchanged wherein the attribute of the structure AA is held containing the version description tags which parenthesize the structure A (62038). For example, since the above normalized entry line "<diff:ver vnum='1'> outputs the result. </diff:ver>" includes a common portion "outputs the result" and difference portion "." extracted separately in the correlation table 67-1, the normalized entry line is replaced by two normalized entry lines "<diff:ver vnum='1'> outputs the result </diff:ver>" and "<diff:ver vnum='1'>.</diff:ver>". Normalized data as shown in Fig. 13 is generated by the above process.

Fig. 17 shows a flowchart of the process performed by the program for updating multi-version documents.

The following description assumes that the contents of a newly revised version V3 is to be added to a multi-version document generated from the versions V1 and V2.

It is assumed that the difference extraction program for structured documents was already performed on both the latest version (in this case, the version V2) included in the multi-version document to be updated and the version to be added (in this case, the version V3), beforehand, to obtain the difference extraction results. It should be noted that Fig. 15 shows data (a tag-attached version V2a) obtained as a result of extracting the contents of the latest version (version V2) including the version description tags incorporated in the multi-version document, and the tag-attached version V2a is used as a comparison target, instead of the version V2 shown in Fig. 4. Fig. 16 shows an example of difference extraction results obtained by comparing V2a with V3. In Fig. 16, each version description tag is the one replaced by a symbol different from that used to indicate each tag included in the original document so as to discriminate one from the other.

Description will be made of the process of updating a multi-version document based on the above difference extraction results.

First, based on information on common portions and difference portions included in the correlation table for the version V2a, a data normalization process is performed on the version V2a included in the multi-version document at step 6302. Next, based on information on common portions included in the correlation table for the version V2a, on each pair of version description tags parenthesizing common portions in the normalized data (6304), a version number of 3 is added to the version attribute and a corresponding divided node ID is added to the node attribute at step 6306. Then, based on information on difference portions included in the correlation table for the version V3, on each difference portion (6308), new difference portion structures are generated at step 6310 by performing steps of: parenthesizing the difference portion with a pair of version description tags; setting its version attribute to the version number "3"; setting its node attribute to the divided node ID; and setting its difference class attribute to the difference type. After that, above new difference structures are inserted at proper positions determined based on relationships between nodes included in the normalized data at step 6312. A multi-version document as shown in Fig. 18 is generated by the above process.

Fig. 19 shows a flowchart of the process performed by the program for outputting a specific version.

First, the program for outputting a specific version receives the version number of a version to be output, and an instruction as to whether version description tags must be output at the time of outputting data at step 6502. The program for outputting a specific version then checks whether the received version number is included in the version attributes of each pair of version description tags in a multi-version document at step 6506, and outputs the structure belonging to each pair of version description tags which includes the received version number, if any, according to the above received instruction (as to whether version description tags must be output). If the received instruction instructs no version description tags to be output, only the internal structure parenthesized by each pair of version description tags which includes the received version number is output at step 6510; otherwise (if version description tags must be output) the structure belonging to each pair of version description tags which includes the received version number is output to a file including the pair of version description tags itself at step 6512. That is, if the instruction requests that no version description tags be output, the original data of each version used to generate the multi-version document is output. If version description tags must be output, on the other hand, each tag-attached version is output.

Fig. 20 shows a flowchart of the process performed by the program for displaying multi-version documents.

First, the program for displaying multi-version documents receives an instruction as to which version must be displayed using which display pattern, at step 6402. This embodiment uses two display patterns: the "display of specified version", in which the contents of a single version are displayed, and the "display of version cross-reference", in which a plurality of versions are displayed at the same time for comparison. The "display of version cross-reference" provides two options: the text cross-reference display, in which version contents are displayed and compared using text, and the tree cross-reference display, in which version contents are displayed and compared using their document trees. Furthermore, the "display of specified version" has display options such as the "difference display" option, which highlights changes made to a specified range of versions such as a range from the version v2 through the version V3, and the "display of modification history" option, which displays the change history of the highlighted changes.

If the "display of specified version" is specified as the display pattern, a specified version is displayed on the screen by referring to the version attribute of each pair of version description tags in a multi-version document in the "display processing of specified version" (6404).

Further, if the "difference display" is selected as an option of the above "display of specified version" with a range of versions specified, the "difference display processing" (6406) highlights the structure belonging to each pair of version description tags whose version attribute satisfies the specified range of versions (that is, the version indicated by the first value of the version attribute is included in the specified range of versions) in the multi-version document to indicate the differences. Still further, if the "display of modification history" is selected to display the change history of the above differences, the "display processing of modification history" (6408) generates a change history list of character stings obtained as a result of extracting the character string parenthesized by each pair of version description tags whose node attribute is the same as that of the pair of version description tags parenthesizing the above differences in the multi-version document, and displays the change history list in a pop-up window. Fig.

21 shows a screen example obtained as a result of selecting the above options "difference display" and "display of modification history".

If the text cross-reference display option of the "display of version cross-reference" is selected as the display pattern, the "display processing of text cross-reference" (6410) obtains matching relationships between nodes in the multi-version document based on the version attribute and the node attribute of each pair of version description tags and displays a screen as shown in Fig. 22 using the obtained matching relationships.

Fig. 22 shows a screen displaying the contents of the version V1 in its display area 221 and the contents of the version V2 in its display area 222. It should be noted that it is possible to employ a display control in which clicking on the character string "fname1" in the display area 221 displays the corresponding character string "str1", which is currently displayed in the display area 222, at the center of a display area 392. If the tree cross-reference display is specified as the display pattern, the "display processing of tree cross-reference" (6412) displays a cross-reference display corresponding to the cross-reference display in Fig. 22, using document trees as shown in Fig. 23.

This completes the description of the first embodiment.

The first embodiment extracts portions common to versions of a document, which are generated each time the document is revised, and adds portions specific to each version to the extracted common portions to generate a multi-version document of the XML format in which the contents of a plurality of versions are efficiently expressed, making it possible to reduce the amount of data necessary for version management, and facilitate exchange of data of a plurality of versions on a network. Furthermore, regarding all structures and text character strings included in a multi-version document, the first embodiment indicates the matching relationships of structures and text character strings belonging to one version with those belonging to another. By obtaining these matching relationships, it is possible to reproduce a given version, highlight changes (difference), and list the structures and text character strings in versions corresponding to a portion (a structure and text character strings) in a target version to which each change was made using the "display of modification history" function, making it easy for the user to grasp changes made to a document and the times when these changes were made.

A second embodiment of the present invention will be described below.

Fig. 24 schematically shows processes employed by the second embodiment. The object of the second embodiment is to generate a multilingual document which expresses a plurality of versions in a single document each written in a different description language but having the same document structure, as represented by an English version whose contents are written in English, and its Japanese version obtained by translating the English version. Since each version has the same document structure, it is possible to match nodes included in one version with those included in another. Therefore, unlike the first embodiment, the second embodiment does not require the difference extraction program for structured documents to obtain the matching relationships between versions. However, the second embodiment has a program 68 for generating a correlation table to generate a correlation table 69, instead.

The process of newly creating a multilingual document according to the second embodiment proceeds as follows. First, the program 68 for generating a correlation table is executed on an English version 40 and its Japanese version 42 to generate the correlation table 69. Then, the program 62 for generating multi-version documents is executed to generate a multi-version document 44 which expresses the contents of both versions in a single file based on the matching relationships stored in the correlation table 69.

Figs. 25 and 26 show examples of the English version 40 and the Japanese version 42, respectively. Figs. 27 and 28, on the other hand, show the correlation tables 69 of the English and the Japanese versions, respectively, each expressing relationships between sentences (text character strings each belonging to a node) included in the respective version. Furthermore, Fig. 29 shows an example of a multilingual document generated based on the above relationships. It should be noted that each pair of version description tags includes a version type attribute "vkind" for identifying each description language, instead. In the figure, each portion parenthesized by a pair of version description tags which includes an expression "vkind='e'" belongs to the English version 40, while each portion parenthesized by a pair of version description tags which includes another expression "vkind='j'" belongs to the Japanese version 42.

The process of adding the contents of a French version 50 written in French and having the same document structure as those of the English and the Japanese versions to the above generated multilingual document (updating the multilingual document) proceeds as follows. After generating a correlation table 692 which stores the relationship between the French version 50 and one of the two versions included in the above multilingual document, the program for updating multi-version documents is executed. This execution generates a multilingual document including the English, the Japanese, and the French versions.

The client 2 obtains a multilingual document stored in the database 7 using a Web browser program 121, and carries out the same display and output controls as those employed by the first embodiment by regarding the obtained multilingual document as a multi-version document.

This completes the description of the second embodiment.

The second embodiment extracts portions common to versions of a document, which are generated each time the document is revised, and adds portions specific to each version to the extracted common portions to generate a multi-version document of the XML format in which the contents of a plurality of versions are efficiently expressed, making it possible to reduce the amount of data necessary for version management, and facilitate exchange of data of a plurality of versions on a network. Furthermore, regarding all structures and text character strings included in a multi-version document, the second embodiment indicates the matching relationships of structures and text character strings belonging to one version with those belonging to another. By obtaining these matching relationships, it is possible to reproduce a given version, highlight changes (difference), and list the structures and text character strings in versions corresponding to a portion (a structure and text character strings) in a target version to which each change was made using the "display of modification history" function, making it easy for the user to grasp changes made to a document and the times when these changes were made.

Since the present invention can display a plurality of versions of a document by showing changes made to each version, the user can easily grasp changes in the contents of the versions.